In [24]:
```
import pandas as pd
import numpy as np
from scipy.stats import norm
import math
import os
import json
```

In [29]:
```
def get_csvs_from_dir(directory):
    datasets_dir = os.path.join(directory, 'environment', 'datasets')
    try:
        csv_files = [os.path.join(datasets_dir, file) for file in os.listdir(datasets_di
        stock_names = [os.path.basename(file).replace('.csv', '') for file in csv_files]
        return csv_files, stock_names
    except FileNotFoundError:
        print(f"The directory {datasets_dir} does not exist.")
        return [], []

def load_data(file_path):
    df = pd.read_csv(file_path)
    df['date'] = pd.to_datetime(df['date'])
    df.set_index('date', inplace=True)
    return df


def black_scholes(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return (S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)) if option_type == 'cal

def calculate_arbitrage(df, r, sigma, T, volume, results_dir, model_name):
    # if model_name is 'lstm', then no profit should be gotten in the first 10 days sinc
    # but the df should still be the same length as the original df
    if model_name == 'lstm':
        df = df[10:]


    results = []
    for index, row in df.iterrows():
        call_price_predicted = black_scholes(row[model_name], row['actual'], T, r, sigma
        put_price_predicted = black_scholes(row[model_name], row['actual'], T, r, sigma,
        call_price_actual = black_scholes(row['actual'], row['actual'], T, r, sigma, 'ca
        put_price_actual = black_scholes(row['actual'], row['actual'], T, r, sigma, 'put
        arbitrage_type, gain = identify_arbitrage(call_price_predicted, put_price_predic
        results.append({'Month': index.strftime('%Y-%m'), 'Arbitrage Type': arbitrage_ty
    results_df = pd.DataFrame(results)

    # if model_name is 'lstm' then first 10 days with 0 value should be added to the res
    if model_name == 'lstm':
        results_df = pd.concat([pd.DataFrame([{'Month': index.strftime('%Y-%m'), 'Arbitr


    # Save results to a csv file
    results_df.to_csv(os.path.join(results_dir, model_name+'_arbitrage.csv'), index=Fals
    return results_df

def identify_arbitrage(call_pred, put_pred, call_act, put_act, volume):
    if call_act > call_pred:
        return 'Buy Call', (call_act - call_pred) * volume
    elif put_act > put_pred:
        return 'Buy Put', (put_act - put_pred) * volume
    return 'No Arbitrage', 0
```

```
def json_serializer(obj):
    if isinstance(obj, (np.int64, np.int32)):
        return int(obj)
    elif isinstance(obj, (np.float64, np.float32)):
        return float(obj)
    elif isinstance(obj, np.ndarray):
        return obj.tolist()
    raise TypeError(f"Type {type(obj)} not serializable")

def update_statistics(arbitrage_df, results_dir, model_name):
    stats = {
        'mean_arbitrage_gain': arbitrage_df['Gain'].mean(),
        'max_arbitrage_gain': arbitrage_df['Gain'].max(),
        'total_arbitrage_gain': arbitrage_df['Gain'].sum(),
        'total_opportunities': len(arbitrage_df),
        'opportunities_with_gain': (arbitrage_df['Gain'] > 0).sum(),
        'opportunities_with_loss': (arbitrage_df['Gain'] < 0).sum(),
        'opportunities_with_no_arbitrage': (arbitrage_df['Arbitrage Type'] == 'No Arbitr
    }

    json_file_path = os.path.join(results_dir, 'profit_stats.json')
    # if file exists, append to it, else create a new file
    if os.path.exists(json_file_path):
        with open(json_file_path, 'r') as file:
            data = json.load(file)
        data[model_name] = stats
        with open(json_file_path, 'w') as file:
            json.dump(data, file, indent=4, default=json_serializer)
    else:
        with open(json_file_path, 'w') as file:
            json.dump({model_name: stats}, file, indent=4, default=json_serializer)
```

In [30]:

```
def process_stocks(directory, r, sigma, T, volume, model_name):
    datasets, stock_names = get_csvs_from_dir(directory)
    stock_names = stock_names
    for stock_name in stock_names:
        results_dir = os.path.join(directory, 'agents', 'trained_models', stock_name)
        predictions_dir = os.path.join(results_dir, 'predictions.csv')
        df = load_data(predictions_dir)
        arbitrage_df = calculate_arbitrage(df, r, sigma, T, volume, results_dir, model_n
        update_statistics(arbitrage_df, results_dir, model_name)
        print(f"Processed {stock_name}")

if __name__ == '__main__':
    project_dir = os.path.dirname(os.getcwd())
    model_name = 'multidqn' # 'arima', 'lstm', 'multidqn'

    # Parameters: risk-free rate, volatility, time to maturity, volume (to simulate larg
    r, sigma, T, volume = 0.01, 0.2, 1/12, 1000000
    process_stocks(project_dir, r, sigma, T, volume, model_name)
```

```
Processed a
Processed aapl
Processed abc
Processed abt
Processed acn
Processed adbe
Processed adi
Processed adm
Processed adp
Processed ads
Processed adsk
Processed aee
Processed aep
Processed aes
```

```
Processed afl
Processed agn
Processed aig
Processed aiv
Processed amzn
Processed jnj
Processed jpm
Processed ko
Processed mmm
Processed msft
```